ISA Dialog Manager

Development Environment

A.06.03.b

This manual provides an overview of the ISA Dialog Manager and its documentation. Furthermore, command line options, environment variables, configuration file, tracing and functions for error analysis are explained.

ISA Informationssysteme GmbH

Meisenweg 33

70771 Leinfelden-Echterdingen

Germany

Notation Conventions

DM will be used as a synonym for Dialog Manager.

The notion of UNIX in general comprises all supported UNIX derivates, otherwise it will be explicitly stated.

< >	to be substituted by the corresponding value
color	keyword
.bgc	attribute
{ }	optional (0 or once)
[ ]	optional (0 or n-times)
<A> | <B>	either <A> or <B>

Description Mode

All keywords are bold and underlined, e.g.

variable **integer** **function**

Indexing of Attributes

Syntax for indexed attributes:

[I]

[I,J] meaning [row, column]

Identifiers

Identifiers have to begin with an uppercase letter or an underline ('_'). The following characters may be uppercase or lowercase letters, digits, or underlines.

Hyphens ('-') are ***not*** permitted as characters for specifying identifiers.

The maximal length of an identifier is *31* characters.

Description of the permitted identifiers in the Backus-Naur form (BNF)

<identifier>	::=	<first character>{<character>}
<first character>	::=	_ | <uppercase>
<character>	::=	_ | <lowercase> | <uppercase> | <digit>

<digit>	::=	1 | 2 | 3 | … 9 | 0
<lowercase>	::=	a | b | c | … x | y | z
<uppercase>	::=	A | B | C | … X | Y | Z

Table of Contents

Notation Conventions ... 3

Table of Contents ... 5

1 Overview ... 7

1.1 Description of Structure and Components ... 7
1.2 Software ... 8
1.3 Documentation ... 8
1.3.1 Attributes, Objects and Resources ... 9
1.3.2 Programming ... 9
1.3.2.1 Built-in Rule Language ... 9
1.3.2.2 Application Programming Interfaces (APIs) ... 10
1.3.2.3 Communication and Integration ... 10
1.3.3 Development Environment ... 11

2 Command Line Options ... 12

2.1 Place Holders in File Names ... 22
2.2 Command Line Options for Setting Code Pages ... 23
2.2.1 Code Page Identifiers ... 23
2.2.2 Encoding Marks for Files ... 23
2.3 Command Line Options in the Simulation Program ... 24

3 Configuration File ... 33

4 Environment Variables ... 35

5 IDM Builder Process ... 36

5.1 Availability ... 36
5.2 The Builder Process Mode ... 36
5.2.1 Example ... 37
5.2.2 Usage Instructions ... 37
5.2.3 Usage Information for the IDM Eclipse Plugin ... 38
5.2.4 Particularities ... 38
5.3 Command Line Options for the Builder Process ... 39

6 Error messages of IDM ... 40

7 Functions for Error Analysis ... 41

7.1 Overview ... 41
7.1.1 Safety Tracing ... 42
7.1.2 Dumpstate ... 42
7.2 Exception Catcher ... 43

8 Dumpstate (Status Information) ... 44

8.1 Process ... 45
8.2 Errors ... 46
8.3 Callstack ... 46
8.4 Events ... 46
8.5 Usage ... 47
8.6 Memory ... 48
8.7 Slots ... 49
8.8 Visible Objects ... 49
8.9 Example ... 50
8.10 Noteworthy for Windows/Threads ... 52

9 Tracing ... 53

9.1 Description of Tracing ... 53
9.2 Configuration of Tracing ... 55
9.3 Time Marks during Tracing ... 58
9.4 Safety Tracing ... 59

Index ... 61

1 Overview

The Dialog Manager is a development tool for the generation of graphic user interfaces. This tool can be used as a system-independent interface between an application and the window system of the relevant target environment.

The Dialog Manager is based on a user interface model that includes

- a presentation layer,
- a dialog layer, and
- an application layer.

All aspects of the presentation layer, i.e. the dialog appearance of the user interface can be manipulated by setting values to object attributes. Please refer to the manuals “Editor”, “Object Reference” and “Attribute Reference” for further details.

The dialog control is defined by the change of attribute values and by calling application functions.

Functions of the application can be called by referencing the function name and the necessary parameters. Please refer to the manuals “C Interface - Functions” and “COBOL Interface” for further details.

1.1 Description of Structure and Components

The DM input file can be divided into two basic data groups.

- The **definition part** contains the definitions for the dialog identifier, resources, default objects, models and objects.
- The **operation rule part** describes the course of the dialog.

The **definition part** has to be structured hierarchically, i.e. only references to already defined objects or resources can be made. For example, if the object "window" is to use the background color "blue", "blue" has to be defined before.

As a consequence, the definitions have to be made in the following order:

- dialog;
- resources;
- default objects;
- models;
- objects;
- rules.

Since rule processing is controlled by the stream of events, no particular structure or order is necessary within the rule definition.

1.2 Software

The standard development environment of the Dialog Manager comprises:

- simulation component and runtime libraries
- interactive WYSIWYG editor
- C and C++ Interface
- XML interface
- Debugger and Profiler
- Tracefile Analyzer (not available for the IDM FOR MOTIF)

The additional options available for the standard Dialog Manager are the following:

- application programming interfaces (APIs)
  - COBOL Interface
  - COBOL Interface for MICRO FOCUS VISUAL COBOL
- communication
  - Distributed Dialog Manager (DDM)
  - OLE interface
- USW Interface to enhance the ISA Dialog Manager with user-supplied widgets
- Eclipse Plugins
  - IDM Eclipse Plugin for creating and editing dialog scripts
  - ISA Makefile Generator

1.3 Documentation

Installation Guide

This guide explains the system requirements for the ISA Dialog Manager and how to install the IDM on different systems.

Release Notes IDM 6
Release Notes IDM 5
Release Notes IDM 4

The release notes report bug fixes, changes and enhancements of each version of the ISA Dialog Manager.

1.3.1 Attributes, Objects and Resources

Attribute Reference

In this manual all predefined attributes of the ISA Dialog Manager objects are described. It contains the definitions and data types of the attributes in the Rule Language and the programming languages C and COBOL.

Object Reference

In this manual all objects of the ISA Dialog Manager are described. For each object its predefined attributes and methods are listed, as well as the events and child objects it supports.

Resource Reference

In this manual all resources of the ISA Dialog Manager are explained. Resources are objects like cursors, colors, texts and fonts, which are used to define certain properties concerning the appearance or behavior of IDM objects.

1.3.2 Programming

1.3.2.1 Built-in Rule Language

Manual Rule Language

This manual explains the Rule Language of the ISA Dialog Manager that is used to program the dynamic behavior of the user interface. Some of the topics are event and rule processing, data types, extent of the Rule Language, syntax of statements, and built-in functions.

Manual Programming Techniques

This manual provides basic techniques for the development of user interfaces with the ISA Dialog Manager. Modularization, use of Models, object-oriented programming and the Datamodel are among the topics covered.

Method Reference

In this manual all predefined methods of the ISA Dialog Manager objects are explained. It comprises the method definitions with their parameters and return values. It is also described, which methods may be overwritten (redefined).

Manual User-defined Attributes and Methods

This manual explains the use of user-defined attributes and methods.

Manual XML Interface

This manual depicts the interface for processing XML (Extensible Markup Language) data with the ISA Dialog Manager. It elucidates the objects available for it and their attributes and methods.

1.3.2.2 Application Programming Interfaces (APIs)

Manual C Interface - Basics

In this manual the basic structure of the API (application programming interface) is depicted that the ISA Dialog Managers offers for applications written in C. The manual describes the data types as well as compiling and linking the applications.

Manual C Interface - Functions

This manual explains all C API (application programming interface) functions of the ISA Dialog Manager. It contains the function definitions with their parameters and return values.

Manual C++ Interface

In this manual the API (application programming interface) of the ISA Dialog Manager for applications developed in C++ is described. It explains the classes to connect to the IDM and their methods. The functioning of the code generator and the code mixer is elucidated.

Manual COBOL Interface

In this manual the API (application programming interface) of the ISA Dialog Manager for applications written in COBOL is described. The data types, all API functions of the COBOL interface, and compiling and linking of the applications are elucidated.

1.3.2.3 Communication and Integration

Manual Distributed Dialog Manager (DDM)

In this manual the network option of the ISA Dialog Manager (Distributed Dialog Manager, DDM) is depicted. With this option distributed applications can be developed where the user interface and the application logic reside on different computers within a network.

Manual User-defined Widgets (USW)

The USW interface (user supplied widgets) is an option of the ISA Dialog Manager to integrate own widgets into the IDM. The manual describes the development of user-defined object classes and their interaction with the IDM.

Manual OLE Interface

This manual describes the OLE interface of the ISA Dialog Manager which is available as an option of the IDM for Microsoft Windows. OLE (Object Linking and Embedding) is a technology for communication between objects and embedding them within each other. The manual explains how OLE clients and servers can be implemented with the IDM.

Manual Automatic Testing and Accessibility

This manual explains features that the ISA Dialog Manager provides to support automatic user interface testing and the development of accessible applications.

1.3.3 Development Environment

Manual Development Environment (this manual)

This manual provides an overview of the ISA Dialog Manager and its documentation. Furthermore, command line options, environment variables, configuration file, tracing and functions for error analysis are explained.

Editor Manual

In this manual the graphical Editor of the ISA Dialog Manager is described with which dialogs can be created and changed interactively.

Debugger Manual

In this manual the Debugger of the ISA Dialog Manager is described. The Debugger is a tool that helps to find errors in dialogs.

Profiler Manual

In this manual the Profiler of the ISA Dialog Manager is described. The Profiler is a tool to analyze the runtime characteristics of dialogs.

Tracefile Analyzer Manual

This manual describes the Tracefile Analyzer of the ISA Dialog Manager, which provides several tools for the analysis of trace files.

2 Command Line Options

The following parameters can be specified in the command line that starts the ISA Dialog Manager:

-IDMbinerror <boolean>

When this binary option is set to *false* (`-IDMbinerror=false`), error messages are suppressed in the binary reading process.

-IDMcallstack <boolean>

This option prevents the IDM from administering a callstack that stores the call list of rules, methods, built-in, application, and interface functions, which is output with the dumpstate.

Default Setting

true

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMcatchexceptions <boolean>

This option prevents the installation of an exception catcher.

Default Setting

true

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMcolor <integer>

This option sets the color variant to be used.

-IDMconfigfile <filename>

When this option is specified, the indicated file is used as configuration file, and the option is removed from the *argv* parameter. When the option is not given, the IDM looks for the environment variable `IDM_CONFIGFILE`. The file specified in this variable is then used as configuration file.

If a configuration file has been specified and the file can be opened, the options given in the file are processed before the remaining options of the command line.

The configuration file contains a list of command line options. The characters BLANK (“ ”), TAB (“\t”) and RETURN (“\n”) can be used as delimiters between options and their parameters.

All options are treated as if they had been input on the command line, which means that they are also used by the application.

Note

-IDMerrfile cannot be used in the same command line.

-IDMconsole

This option causes log file and error messages to be written to the standard output channels (STDOUT) on MICROSOFT WINDOWS – analog to Unix. If the process does not run within the console, a new console is opened for potential outputs. In the predecessor versions of Windows XP, a new console is **always** opened.

Availability

Versions since A.05.02.f; platform Microsoft Windows

-IDMcp_appl <code page>
-IDMcp_format <code page>
-IDMcp_input <code page>
-IDMcp_io <code page>
-IDMcp_output <code page>

These are options to set the code page for various operations. See chapter “Command Line Options for Setting Code Pages”.

-IDMcursor <integer>

This option sets the cursor variant to be used.

-IDMdumpstate <enum>

This option influences the output of IDM status information (dumpstate). Via the <*enum*> parameter the output of certain information is enforced.

Parameters

Value (enum)	Meaning
dump_all	All sections are written out in an abbreviated form. This corresponds to the output in case of a FATAL ERROR.
dump_error	The sections ERRORS, CALLSTACK and EVENTS are written out in an abbreviated form. This is the normal output in the case of EVAL ERRORS.
dump_events	The section THISEVENT/EVENT QUEUE is written out in full.
dump_full	All sections are written out in full.

Value (enum)	Meaning
dump_locked	The section SLOTS is written out in full. In addition, for locked objects their attribute values are written out.
dump_memory	The section MEMORY is written out in full.
dump_none	No action (nothing is written out).
dump_process	The section PROCESS is written out in full.
dump_short	All sections (excluding SLOTS) are written out in an abbreviated form.
dump_slots	The section SLOTS is written out in full.
dump_stack	The section CALLSTACK is written out in full.
dump_usage	The section USAGE is written out in full.
dump_uservisible	The section VISIBLE OBJECTS is written out in full for all visible top-level objects including their children, the pre-defined and user-defined attributes.
dump_visible	The section VISIBLE OBJECTS is completely written out.

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMdumpstateseverity <string>

The output of a dumpstate normally occurs when an EVAL ERROR or a FATAL ERROR arises. With this option, a dumpstate output can be enforced for other types of errors and messages.

Parameters

First Character	Dumpstate Output…
E	in case of normal error messages [E: …]
F	in case of fatal error messages [F: …]
I	in case of informative messages [I: …], warnings and error messages
O	only in case of EVAL ERROR
W	for warnings [W: …] and error messages

Default Setting

“F”

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMenv <variable>=<value>

The ISA Dialog Manager allows setting environment variables from the command line. This is especially useful with the IDM on Microsoft Windows. Microsoft Windows generally provides no means to set environment variables for all applications.

With the command line option **-IDMenv <variable>=<value>** those variables can be set for an IDM application. In doing so, the environment variables of Microsoft Windows are overwritten for the IDM application. The environment variables are set only temporarily and only for the IDM application.

Note

No blanks are allowed between the variable name, the equals sign and the variable value.

This option can be indicated several times in order to set different variables.

Example

Windows

```
idm -IDMenv BINARY=c:\idm\bin -IDMenv IF=c:\idm\if dialog.dlg
```

Unix

```
idm -IDMenv binary=/home/user1/bin -IDMenv IF=/usr/idm/interface
dialog.dlg
```

-IDMerrfile <filepath>|none

This option specifies an error file. The file traces all errors that occur while the application runs.

The same can be achieved by setting the environment variable `IDM_LOGFILE` to a file path.

The command line option overwrites the environment variable when both are set.

With the command line option and the environment variable, *none* can be specified instead of a file path to prevent the creation of an error file.

In the file name, place holders can be used (see chapter “Place Holders in File Names”).

Note

When the option **-IDMtracefile** (or the respective environment variable) is used in coincidence with this option, the error file specified with this option may not be created or may be left empty. Possible error messages then have been written to the trace file.

-IDMerrwinfile <filepath>|none

This options redirects error messages, which are displayed as messageboxes or output to the console, to the specified file. This enables for example server processes (especially on Microsoft

Windows) to run without any user interaction (like the confirmation of error dialogs).

The same can be achieved by setting the environment variable `IDM_ERRWIN` to a file path.

The command line option overwrites the environment variable when both are set.

With the command line option and the environment variable, *none* can be specified instead of a file path. This prevents the display of error dialogs and the output of error messages to the console, as well as the creation of a file to trace the errors.

In the file name, place holders can be used (see chapter "Place Holders in File Names").

Note

When the options **-IDMerrfile** or **-IDMtracefile** (or the respective environment variables) are used in coincidence with this option, the file specified with this option may not be created or may be left empty. Possible error messages then have been written to file specified with the other option.

-IDMfatalneterrors <boolean>

With `-IDMfatalneterrors true` a compatible behavior to the IDM versions before A.05.01.d is set for the DISTRIBUTED DIALOG MANAGERS (DDM), enforcing an immediate termination on client and server side when a network, protocol or version error occurs. This means that except for local applications no more *start* and *finish* events will be triggered and **AppFinish** will not be called.

When using a command line option is impossible, the option *DMF_FatalNetErrors* of the function **DM_Initialize** can be utilized instead of **-IDMfatalneterros**. *DMF_FatalNetErrors* takes precedence over **-IDMfatalneterrors**.

See Also

C function DM_Initialize

-IDMfont <integer>

This option sets the font variant to be used.

-IDMformat <integer>

This option defines, which variant of the format resources is used.

-IDMindent <indent>[:<tabsize>[:<traceindent>]]

This option controls the indenting in source code and optionally for tracing. Moreover, the replacement of spaces through tabs can be configured.

For <*indent*>, which defines the indenting of source code, and for the optional <*traceindent*>, which sets up the indenting within the trace file, integral numbers >=*0* are allowed. For the optional <*tabsize*> that controls the replacement of spaces through tabs, only the values *0* and *8* are allowed.

For instance, to get an indenting of three characters for each level without tab replacement, you specify *"3:0"*; an indenting of one tab for each level is achieved with *"8:8"*.

Note

For the trace file, the indenting can also be set up on the ***setup*** object.

-IDMkeyboard <integer>

This option sets the accelerator variant to be used.

-IDMlanguage <integer>

This option defines, which variant of the text resources is used.

-IDMno_yi_monitoring

With this option, the calling of monitoring functions, which have been registered with **YiRegisterUserEventMonitor**, can be disabled. This option supports error analysis in the case where a monitoring function is supect to cause the error.

The environment variable IDM_NO_YI_MONITORING can be used instead of this command line option.

The command line option overwrites the environment variable when both are set.

Alternatively, the option *.options[opt_yi_monitoring]* of the ***setup*** object can be set to *false*. If monitoring functions have been disabled through the command line option or the environment variable, they cannot be enabled by setting *.options[opt_yi_monitoring] = true* on the ***setup*** object.

-IDMobjdump_fkey <func_key_no>

This command line option effects that the source code of the active window is written to the trace file when the function key with the number *<func_key_no>* is pressed. In the trace file, the output source code is marked with the trace codes *“DC”* and *“DR”*.

-IDMscale <integer>

The **-IDMscale** option determines the scaling with which the application is to be displayed. A value of *0* switches off the scaling. The value is given in %.

Default setting:

The current scaling used by the system.

Attention

It is not recommended to use a scaling > 0 and < 100%, as this may impact the display and operation of objects.

Limitations WINDOWS:

This startup option can only be used to turn DPI awareness on or off. However, this option should not be used under Microsoft Windows. Since DPI awareness is a property of the application, it should only be specified via a manifest file. Using the startup option under Windows results in a warning in the trace or log file.
For test purposes, DPI awareness can be switched on (value: *1*) or off (value: *0*), but should never be changed while the application is running.

Limitations Qt:

Whether a scaling factor set here is actually applied depends strongly on the desktop environment and its support for HighDPI.

Availability

Since IDM version A.06.03.a

See alsochaper“HighDPI UnterstützungSupport”in manual“Programming Techniques”

-IDMsearchpath <search path>

This option sets the search path in which dialog, module, interface and binary files for imports with **use** are searched. This option overrides the search path, which can also be set by specifying the environment variable `IDM_SEARCHPATH`.

The search path is a semicolon-separated list of paths (absolute or relative) with the following special features:

~ or ~:	Search beneath the directory in which the application is located.
"" (empty path)	Search in the current working directory (same behavior as in previous versions).
<ENVNAME>:	Search in the paths that are defined in the environment variable `<ENVNAME>`.

Note

The search path can also be set with **DM_ControlEx()** and the ***setup*** object.

Availability

Since IDM version A.06.02.g

See also

Chapter “Search Path for Interface, Module, Dialog, and Binary Files” in manual “Programming Techniques”

-IDMserver

This option queries the version of the window system interface.

This option only works with the Motif version of the ISA Dialog Manager. With the Windows version it produces an error.

-IDMshowerror

Outputs internal errors, which are handled by IDM, into the trace file too. Normally these errors can be ignored. However, they may provide hints for the cause of other errors in some situations.

-IDMsource <integer>

This option defines, which variant of the source resources is used for drag&drop actions.

-IDMstrace

The trace file is used in safety mode. For this option to work, the tracing must be activated via the **-IDMtracefile <filepath>** option.

In order to keep the trace file running during long application sessions without experiencing a slower running system and the use of too many resources, safety tracing uses a limited ring buffer that is held in the memory. The content of the ring buffer is saved to the file on termination of the application.

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMstracefile <filepath>

This option is the short form for the combination of the options **-IDMtracefile <filepath>** for activating the tracing and **-IDMstrace** for setting the safety mode.

In the file name, place holders can be used (see chapter "Place Holders in File Names").

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMstraceopts <string>

This option activates the safety tracing and at the same time defines the settings for this. For this option to work, the tracing must be switched on via the **-IDMtracefile <filepath>** option.

The string parameter influences the setting as follows:

Template	Setting
c<integer>	Bytes per line
h	Hierarchical output: Retention of as many hierarchical levels as possible
l<integer>	Number of lines
r	Rotating output (default): The oldest line is replaced by the newest
s<integer>	Length limit for strings

Through concatenation it is possible to carry out numerous settings at the same time. The order is arbitrary.

Instead of this option the environment variable IDM_STRACEOPTS can be used. The command line option overwrites the environment variable when both are set.

Example

Safety tracing with *300* lines and *80* bytes per line is activated via the option **-IDMstraceopts l300c80**.

Availability

IDM versions A.05.01.g3, A.05.01.h, since A.05.02.e

-IDMtarget <integer>

This option defines, which variant of the target resources is used for drag&drop actions.

-IDMtile <integer>

This option sets the tile variant to be used.

-IDMtiledpi <integer>

This startup option specifies the DPI value to be applied to tiles. This DPI value plays a role if images were created for a specific DPI value and are to be displayed accordingly. By default, *96* DPI is assumed.

Availability

Since IDM version A.06.03.a

See alsochaper“HighDPI UnterstützungSupport”in manual“Programming Techniques”

-IDMtracefile <filepath>

This option activates the tracing, which logs all function calls from the IDM, all calls from the IDM to the application and all executed rules into the given file (trace file, see chapter “Tracing” for further information).

The same can be achieved by setting the environment variable `IDM_TRACEFILE` to a file path.

The command line option overwrites the environment variable when both are set.

In the file name, place holders can be used (see chapter “Place Holders in File Names”).

-IDMtracetime <integer>

This option writes additional timestamps into the trace file , in which all IDM function calls, all calls from the IDM to the application, and all executed rules are logged. It is thus possible to perceive the absolute or relative time needed for functions or rules and to take tuning measures if necessary.

Value range

0

No times are logged in the trace file.

1

This value indicates start time mode. In this mode all start and end times are logged. The time needed for a single structure may then be calculated with the difference. In this mode only the system and user time will be considered.
The times are given in format [hh:mm:ss:uuu] at the beginning of line:

- hh = hours
- mm = minutes
- ss = seconds
- uuu = milliseconds

2

This value indicates the trace time mode. In this mode the time difference to the last logged call is given. It is thus possible to easily recognize how much time is needed for individual actions.
In this mode the time difference to the last trace output is given in the format [sss:uuu] at the beginning of line:

- ss = seconds
- uuu = milliseconds

3

This value specifies the real-time mode. In this case the real time is indicated for each action to be logged in the trace file.
In this mode the real time is given in format [hh:mm:ss] at the beginning of line:

- hh = hours
- mm = minutes
- ss = seconds

-IDMusepathmodifier <string>

This option controls the conversion of a Use Path into a file name. It allows to control file name conversion to upper and lower case. This option should only be used with caution.

The following options are available:

L The entire file path is converted to lower case.
The Use Path `Base.Colors` becomes the file names*base/colors.if*, *base/colors.mod* and *base/colors.bin*.

U The entire file path including the extension is converted to upper case.
The Use Path `Base.Colors` becomes the file names *BASE/COLORS.IF*, *BASE/COLORS.MOD* and *BASE/COLORS.BIN*.

u The entire file path except for the extension is converted to upper case.
The Use Path `Base.Colors` becomes the file names *BASE/COLORS.if*, *BASE/COLORS.if* and *BASE/COLORS.if*.

l Only the first letter of each file path segments is converted to lower case.
The Use Path `BaseModels.MWin` becomes the file names *baseModels/mWin.if*, *baseModels/mWin.mod* and *baseModels/mWin.bin*.

Availability

Since IDM version A.06.02.g

-IDMversion

This option queries the version of the IDM runtime environment in use.

The version is output with the pattern

k.vn.sv.pl<additional_info>.

The meaning of the parts is:

k	Kind of release (capital letter, “A” in most cases)
vn	Version number, major release (two digits)
sv	Sub-version number, minor release (two digits)
pl	Patch level (lowercase letter and optionally one digit)
<additional_info>	for internal purposes only

2.1 Place Holders in File Names

When it is stated for command line options or environment variables, that place holders can be used in the file name, these are the available place holders:

%Y	year
%O	month
%D	day
%H	hour
%M	minute
%S	second
%J	day of year
%C	unique number
%A	application name
%T	tty ID
%U	user ID
%P	process ID

The place holders are substituted through the respective values at runtime.

2.2 Command Line Options for Setting Code Pages

The code page options allow to set the character encoding that IDM uses for different operations. The following command line options are available, in which <code page> may be replaced by the code page identifiers listed in chapter “Code Page Identifiers”.

Option	Description
-IDMcp_appl <code page>	Defines the code page for strings received or returned by application functions. The better solution however is to set this code page through DM_Control from within the application.
-IDMcp_format <code page>	Defines the code page for strings received or returned by format functions. The better solution however is to set this code page through DM_Control from within the application.
-IDMcp_input <code page>	Defines the code page which IDM uses for interpreting dialog-, module-, interface-, and init-files if the files carry no encoding mark (see chapter “Encoding Marks for Files”).
-IDMcp_io <code page>	Abbreviation for setting input and output code page at once (**-IDMcp_input** and **-IDMcp_output**).
-IDMcp_output <code page>	Defines the code page which IDM uses for output files. The option applies to dialogs, modules, interfaces, init-, log-, and trace-files as well as stdout and stdin.

2.2.1 Code Page Identifiers

With the command line options for setting code pages the identifiers listed below may be used for the parameter <code page>:

acp	ascii	cp437	cp850	cp1252
dec169	hp15	iso6937	iso8859	no_conv
utf8	utf16	utf16b	utf16l	winansi

2.2.2 Encoding Marks for Files

Text files can contain encoding marks within their first 8 bytes that indicate the character encoding of the files. Is an encoding mark is present it takes precedence over a command line option for IDM.

These are the encoding marks recognized by IDM:

Encoding mark	Character encoding
Byte sequence 0xfeff	UTF-16 BE (big-endian)
Byte sequence 0xffef	UTF-16 LE (little-endian)
// UTF8 or // utf8	UTF-8
// 8859	ISO-8859-1
// 1252	CP1252

The hexadecimal byte sequences “feff” and “ffef” are called “Byte Order Marks” (BOM) and are defined in the Unicode standard. The define the sequence of bytes within character codes. “feff” indicate “big-endian” order where bytes with higher numeric significance come first, “ffef” indicates “little-endian” order, where bytes with lower numeric significance come first.

If the output code page is set to UTF-8, UTF-16, ISO-8859, or CP1252 through **-IDMcp_output** or **-IDMcp_io**, IDM writes the appropriate encoding mark as prefix into dialogs, modules and interfaces.

Note

Text editors normally do not display the byte order marks “feff” or “ffef”, but they are visible when opening files with a hex editor.

2.3 Command Line Options in the Simulation Program

The following options can only be used in the **simulation program**:

+application <applicationID>

This option is only valid with the option **+writeproto**!

When giving this option, the function prototypes of the functions included in the object *applicationID* are written out.

-bindir <directory path>

This option allows the definition of the target directory where the binary files (with the file extension *.bin*) are created. By default they are created in the working directory.

Die Option kann nur zusammen mit den Startoptionen **-compile**, **-compile1**, **-recompile** und **-recompile1** verwendet werden.

The option can only be used in combination with the command line options **-compile**, **-compile1**, **-recompile** and **-recompile1**.

+/-builder

This option causes the IDM and PIDM to run in the builder mode. If it has not already happened, it is started as IDM builder process in the background. All requests for writing interface, binary, funcmap or trampoline files are passed to this IDM builder process. This process works in the **shared**

modules mode to save time when reloading imported modules.

With **-builder**, the server and client mode is determined by the actions (**-writebin**, **-writeexport**…) while **+builder** explicitly activates the client mode. For **-builder**, client mode is automatically assumed when it is used with an action. Without an action it works in the builder process (server).

Availability

Versions since A.05.02.f; platforms MICROSOFT WINDOWS, UNIX/LINUX

-builderid <builder-identifier>

Normally the IDM builder process and the IDM or PIDM, that use the process in the builder mode, have the same father process. This can be bypassed by specifying an own builder ID as a file name **without** the preceding path.

Availability

Versions since A.05.02.f; platforms MICROSOFT WINDOWS, UNIX/LINUX

-builderstop

With this option, a running IDM builder process can be stopped before reaching its time-out period.

Availability

Versions since A.05.02.f; platforms MICROSOFT WINDOWS, UNIX/LINUX

-buildertimeout <secs>

With this option, the waiting time of the IDM or the PIDM in the builder process mode is defined in seconds. After this time has been reached, the IDM builder process terminates. After the defined waiting period, connection attempts and calls fail.

Note for UNIX/LINUX

If a too small **builder timeout** is chosen, this may result in no communication at all between client and server. The **pipe** file (usually found under */tmp/pipe_idmbuilder…*) that is created as a result and not cleared by the system may then have to be deleted manually.

Availability

Versions since A.05.02.f; platforms MICROSOFT WINDOWS, UNIX/LINUX

-classname <classname>

This option limits the writing of class definitions of USW classes for the IDM ECLIPSE PLUGIN to the specified class.

It is only valid with the option **-writeclassdef**.

-cleancompile
-cleancompile1

This option deletes all interface and binary files for the loaded dialog and module files as well as all submodules imported with **use**.

Die Option **-cleancompile1** führt die Aktion lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

The option **-cleancompile1** performs the action only for the loaded dialog or module and **not** for submodules imported per **use**.

Availability

Since IDM version A.06.02.g

See also

Chapter "Compiling Interface and Binary Files for Imports with use" in manual "Programming Techniques"

-cobbasename <basefilename>

Sets the base name used in the COPY statement of the generated COBOL file. This option overrides the value that is computed from the **-cpyname**, **-cobname** or **-writetrampolin** options.

Can only be used together with the option **+/-writetrampolin**.

Availability

Since IDM version A.06.02.g

-cobname <basefilename>

Sets the base name for the generated COBOL file. This option overrides the value that is computed from the name specified with **-writetrampolin**.

The file extension of this file is ".cbl" if one of the options **-ufcob**, **-mfviscob** or **-mfviscob-u** is given, otherwise ".cob".

Can only be used together with the option **+/-writetrampolin**.

Availability

Since IDM version A.06.02.g

-compile
-compile1

This option causes the interface and binary files to be written for the loaded module or dialog and for all loaded submodules imported per **use**.

However, these are only rewritten if the file date of the source file(s) is newer than the interface and binary files. This option is intended for use in a Makefile.

Um die Zieldateien auf jeden Fall, ohne Prüfung des Dateidatums, neu zu erzeugen, kann die Option **-recompile** verwendet werden.

To recreate the target files in any case, without checking the file date, the **-recompile** option can be used.

Possibly necessary directories resulting from the Use Path are created if required. The output directories of the target files can be controlled using the **-ifdir** and **-bindir** options.

Since the generated interface and binary files have predefined file extensions (*.if* and *.bin*), it is possible to create these files mixed with the sources.

Attention

Existing files will be overwritten.

Die Option **-compile1** führt die Aktion lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

The option **-compile1** performs the action only for the loaded dialog or module and **not** for sub-modules imported per **use**.

Availability

Since IDM version A.06.02.g

See also

Chapter "Compiling Interface and Binary Files for Imports with use" in manual "Programming Techniques"

-cpyname <basefilename>

Sets the base name for the generated COBOL copy file. This option overrides the name that is given by **-cobname** or **-writetrampolin**. The file extension of this file is ".cpy".

Can only be used together with the option **+/-writetrampolin**.

Availability

Since IDM version A.06.02.g

-ifdir <directory path>

This option allows the definition of the target directory where the interface files (with the file extension *.if*) are created. By default they are created in the working directory.

Die Option kann nur zusammen mit den Startoptionen **-compile**, **-compile1**, **-recompile** und **-recompile1** verwendet werden.

The option can only be used in combination with the command line options **-compile**, **-compile1**, **-recompile** and **-recompile1**.

-mfviscob

Generates COBOL copy files for MICRO FOCUS VISUAL COBOL.

Can only be used together with the options **+writeheader** and **+/-writetrampolin**.

Availability

COBOL Interface for MICRO FOCUS VISUAL COBOL only.

-mfviscob-u

Generates COBOL copy files for Micro Focus Visual COBOL with support for *National Character* (Unicode, UTF-16).

Can only be used together with the options **+writeheader** and **+/-writetrampolin**.

Availability

COBOL Interface for Micro Focus Visual COBOL only.

-noif

This option suppresses the generation of interface files.

Die Option kann nur zusammen mit den Startoptionen **-compile**, **-compile1**, **-recompile** und **-recompile1** verwendet werden.

The option can only be used in combination with the command line options **-compile**, **-compile1**, **-recompile** and **-recompile1**.

+/-profile <filename>

With help of this option a configuration file is loaded.

-profile	configuration file is loaded before **DM_StartDialog**
+profile	configuration file is loaded after **DM_StartDialog**

See Also

C interface function DM_LoadProfile

COBOL interface function DMcob_LoadProfile

-recompile
-recompile1

Diese Option bewirkt, dass für das geladene Modul bzw. den geladenen Dialog sowie alle per **use** importierten, geladenen Untermodule die Interface- und Binärdateien geschrieben werden.

This option causes the interface and binary files to be written for the loaded module or dialog and for all loaded submodules imported per **use**.

Anders als bei der Option **-compile** werden die Zieldateien **immer** neu erzeugt. Es findet keine Datumsprüfung statt.

Unlike the option **-compile**, the target files are **always** regenerated. There is no date check.

Eventuell notwendige Verzeichnisse, die sich aus dem Use-Pfad ergeben, werden wenn erforderlich angelegt. Die Ausgabeverzeichnisse der Zieldateien können über die Optionen **-ifdir** und **-bindir** gesteuert werden.

Possibly necessary directories resulting from the Use Path are created if required. The output directories of the target files can be controlled using the **-ifdir** and **-bindir** options.

Da die erzeugten Interface- und Binärdateien vorgegebene Dateiendungen (*.if* und *.bin*) besitzen, ist die Erzeugung dieser Dateien gemischt mit den Quellen möglich.

Since the generated interface and binary files have predefined file extensions (*.if* and *.bin*), it is possible to create these files mixed with the sources.

Attention

Vorhandene Dateien werden überschrieben.

Existing files will be overwritten.

Die Option **-recompile1** führt die Aktion lediglich für den geladen Dialog bzw. das geladenen Modul und **nicht** für per **use** importierte Untermodule aus.

The option **-recompile1** performs the action only for the loaded dialog or module and **not** for submodules imported per **use**.

Availability

Since IDM version A.06.02.g

See also

Chapter "Compiling Interface and Binary Files for Imports with use" in manual "Programming Techniques"

+searchsymbol

A module is searched in a path which is defined via an environment variable. If the corresponding file is found in a directory, it will be loaded and the further search will be interrupted.

In order for this environment variable to be considered on loading the modules, the following option has to be indicated additionally when creating the interface file:**+searchsymbol IDMLIB**. All file names will then be prefixed by the symbol "IDMLIB".

General Syntax

```
idm +writeexport <export file name> +searchsymbol <environment variable>
  <module name>
```

Example

```
idm +writeexport color.if +searchsymbol IDMLIB color.mod
```

See Also

Chapter "Modularization" in manual "Programming Techniques"

-ufcob

Generates COBOL copy files for MICRO FOCUS COBOL.

Can only be used together with the options **+writeheader** and **+/-writetrampolin**.

-userregistry

This option can be specified in addition to **-writeole** to register an OLE control for the current user only. The registration data is then written to the Windows Registry under HKEY_CURRENT_USER.

Availability

Since IDM version A.06.01.g

-writebin <binaryfile>

A binary file is created from the ASCII dialog file when this option is specified.

The binary file has a significantly shorter load time, and it cannot be edited by the end user.

-writeclassdef <xmlfile>

With this option the class definitions of all registered USW classes are written to the specified file, which can be used with the IDM ECLIPSE PLUGIN. The class definitions are required by the IDM ECLIPSE PLUGIN, e.g to enable content assist to propose the attributes for USW classes.

+/-writedialog <filename>

With this option a dialog or module in its current state can be written as text file to “filename”. The character encoding (code page) of the file can be set through **-IDMcp_output** or **-IDMcp_io** (see chapter “Command Line Options for Setting Code Pages”). Without these options, the dialog or module will be ASCII encoded.

+writeexport <interfacefile>

With this option the developer can create the interface file out of her/his module. Comments which are prefixed by !! and which are attached before the actual objects are included in the interface file. In doing so, the commenting of dialog sources can be made available also to the user of a module.

See Also

Chapter “Modularization” in manual “Programming Techniques”

+writefuncmap <basefilename>

This option creates a C file and an include file. The include file contains the function prototypes for all functions declared in the DM dialog file (see also option **+writeproto**). A function table is put in the C file so that, to link the functions, the contained function **BindFunctions_<name of dialog>** has to be called.

See Also

Manual “C Interface - Basics”

+writeheader <basefilename>

The command line option **+writeheader** may be used to generate prototypes and record definitions for dynamically connected functions.

The command line for the simulation is built like this:

```
pidm [+application <name of application>] +writeheader <basefilename>
  <dialogfile>
```

If there are C functions, a header file with the suffix **.h** is created that contains the appropriate function types and record definitions. If COBOL functions with record parameters exist, the copy sections are written to the file with the extension **.cpy**. If no respective functions are present, no file will be generated.

The command line option is sort of a mixture between the options **+writefuncmap** and **+/-writettrampolin**, but without the generation of C and COBOL code. Therefore it is only suitable for the dynamic binding of application functions.

Availability

Since IDM version A.06.01.d

-writeole <basefilename>

With this option, the ISA Dialog Manager generates the necessary files to register an OLE server with the system. An **idl** file for creating the type library and a **reg** file with the registry entries are generated. The registry entries are stored under HKEY_LOCAL_MACHINE in the Windows Registry so that the registration is effective for all users.

To register an OLE control for the current user only, you can additionally specify the option -userregistry.

Note

The Windows tool **regedit.exe** usually sets no error status in case of an error. If there are problems with the start of an OLE control, the registry should be checked to see if the registry data was loaded at all. Changes to the Windows Registry should be made with caution, as Windows may fail to start in case of errors.

See Also

Chapter “Generating Interface Information” in manual “OLE Interface”

+writeproto <filename>

This option creates a file that contains function prototypes for all function declared in the DM dialog file. This file can be used as C include file for the application to check the correctness of the function call.

+writerefs <filename>

By means of this option you can check for any dialog whether all defined models are really used within this dialog.

If a model is not used, i.e. it has 0 references, it can be possible that this model is used in a rule to create objects dynamically. Therefore, the output of this option should be used as a hint for a used model.

Example

```
idm +writerefs TestRefs.txt Test.dlg
```

+/-writetrampolin <basefilename>

To provide functions for dialogs including records with an application, the C modules generated by the DM have to be compiled and linked. The modules are generated by calling the simulation via the option **+/-writetrampolin**:

```
idm +writetrampolin <outfile> <dialogscript>
```

This statement generates the necessary modules from a dialog script in order to call the functions. The corresponding header files (C and/or COBOL) are created according to the kind of functions using such records.

In combination with **+/-writetrampolin** the following options can be used:

- -ufcob, -mfviscob and -mfviscob-u to create copy files for MICRO FOCUS COBOL or MICRO FOCUS VISUAL COBOL respectively.
- **-cobbasename**, **-cobname** and **-cpyname** to customize the base names of the created files.

3 Configuration File

Configurable variables and records can be redefined by the user in the configuration file.

The configuration file can be loaded with DM_LoadProfile.

Within the configuration file, any number of entries of the following scheme are valid:

Syntax

```
<variable> := <value> ;
<record> . <attribute> { [<index>] } := <value> ;

// <comment up to end of line>
/* <comment, multi-line too> */
```

<variable> has to be indicated as configurable with **config**.

At <record>, <attribute> has to be *.configurable true*, and the wished attribute has to be available as DM-internal or user-defined attribute.

These attributes can optionally also be addressed indexed. Then, however, they have to be defined in the corresponding record.

As <value>, all constant values are permitted which can be used when defining variables with initialization and when initializing user-defined attributes. The corresponding values' data types have of course to match the variables' and the record attributes' data types.

Example

Dialog Script

```
dialog Example

config variable string Name := "Jack";

color C1 "red";
color C2 "green";

model window WinMod
{
  .bgc C1;
}

record Settings
{
  .configurable true;

  boolean Mode[3];
```

```
    .Mode[1] := true;
    .Mode[2] := false;
    .Mode[3] := false;

  object Windowcolor shadows WinMod.bgc;
}
```

Configuration File

```
// configuring a variable
Name := "Peter";
// configuring a record
Settings.Mode[2] := true
// the model is configured directly with shadows
Settings.Windowcolor := C2;
```

4 Environment Variables

The external information should always be loaded by means of environment variables from the Rule Language as well as from the programming interface (see also functions DM_LoadDialog or DM_ LoadProfile in manual “C Interface - Functions”). These variables are:

- external pictures of a tile
- interface and module descriptions
- dialog files
- profiles

The indicated name may have the following structure (the same applies to all file accesses):

```
<environment variable>:<name of dialog file>
```

whereby the environment variable may be an arbitrary path under which the dialog file shall be searched for. This path is checked on loading and the first data which is found will be loaded.

This can easily be configured on the target computer.

Example

Rule Language:

```
tile Checkbox "IDM_IMAGEPATH:Check.gif";
```

Before the program start the environment variable `IDM_IMAGEPATH` has to be set to the directory in which the GIF file is located.

- Microsoft Windows

  ```
  set IDM_IMAGEPATH =d:\appl\images
  ```

- Unix

  ```
  setenv IDM_IMAGEPATH /usr/local/appl/images
  ```

5 IDM Builder Process

5.1 Availability

- Platforms: The builder process mode is only available for MICROSOFT WINDOWS and UNIX respectively LINUX, but not for VMS.
- Versions: since A.05.02.f

5.2 The Builder Process Mode

For complex, modular IDM applications, the time requirement for the creation of the application with **make** or other build management tools are significantly determined by the loading times of the modules and dialogs.

Every call of the simulation (IDM or PIDM) for the creation of an interface, binary, funcmap or trampoline file necessarily loads the specified dialog or module file. If the application consists of several modules with many deep import dependencies, long loading times are primarily caused by reloading the imported modules.

A clear reduction of this reloading time can be achieved by using the IDM or the PIDM in the **builder process mode**.

Here an IDM simulation program runs in the background in the builder process mode as **server** and waits for requests to create interface, binary, funcmap and trampoline files. Hereby, this IDM builder process runs in the **shared modules mode** which is normally activated by setting the environment variable DM_SHARED_MODULES. In this process, the reuse of imported modules is based on the equality of the interface name and not, as is usually the case, on the import descriptor and the superordinate import. This leads to a clear reduction of loading time for modules with many imports.

The IDM or PIDM calls required during the creation of the application merely connect as **client** to the IDM builder process. The necessary arguments are sent to the server and the return status is passed on.

The IDM builder process is not started manually, but starts automatically when correctly used (option **+builder**). After a definable time-out period (option: **-buildertimeout <secs>**) it stops automatically. The process may also be stopped manually before the time-out period ends (option **-builderstop**).

With this, the previous use of the IDM or the PIDM should be converted to the use in the builder process mode with relative ease and without any problems.

5.2.1 Example

Makefile With a Simple Modularized Dialog

```
PIDM=pidm
IDMARGS=-IDMenv MODPATH=if;.
IDMARGS_WRITEBIN=$(IDM_ARGS) +searchsymbol MODPATH –writebin
IDMARGS_WRITEIF=$(IDM_ARGS) –writeexport
all:: bin/dialog.dlg
bin/dialog.dlg: dialog.dlg if/defaults.if
    $(PIDM) $(IDMARGS_WRITEBIN) $@ dialog.dlg
if/defaults.if: defaults.mod
    $(PIDM) $(IDMARGS_WRITEIF) $@ dialog.dlg
bin/defaults.mod: defaults.mod
    $(PIDM) $(IDMARGS_WRITEBIN) $@ defaults.mod
```

Converted Makefile

The activation of the builder process mode is underlined

```
PIDM=pidm
IDMARGS=-IDMenv MODPATH=if;. +builder
# Windows:
# IDMARGS=-IDMconsole -IDMenv MODPATH=if:. +builder
IDMARGS_WRITEBIN=$(IDM_ARGS) +searchsymbol MODPATH –writebin
IDMARGS_WRITEIF=$(IDM_ARGS) –writeexport
all:: bin/dialog.dlg builderstop
bin/dialog.dlg: dialog.dlg if/defaults.if
    $(PIDM) $(IDMARGS_WRITEBIN) $@ dialog.dlg
if/defaults.if: defaults.mod
    $(PIDM) $(IDMARGS_WRITEIF) $@ dialog.dlg
bin/defaults.mod: defaults.mod
    $(PIDM) $(IDMARGS_WRITEBIN) $@ defaults.mod
builderstop::                      # optional
    $(PIDM) –builderstop
```

5.2.2 Usage Instructions

For the reuse of the same IDM builder process (i.e. the same IDM or PIDM in the builder process server mode), the process number of the father process of the IDM or PIDM client must be identical. Otherwise, the option **-builderid** can be used and the IDM builder process can be manually started and stopped. It is advisable that IDM specific construction steps take place successively to enable the reuse of the IDM builder process within the adjustable time-out period.

No IDM builder process runs initially. It is automatically started with the option **+builder** and contains the environment of the process being started. During the communication between the builder client and the server, the environment variables and the work directory of the builder client set with **-IDMenv** are passed on. These variables therefore overlay the variable set in the IDM builder process. Changing the work directory ensures an identical situation for client and server. If no connection can be established between client and server, this leads to the typical error messages such as *"… can't open/connect builder pipe …"*.

The IDM builder process is not intended to parallelize the build. Only one client at a time can connect to the IDM builder process.

If the maximum number of modules has been reached (ca. 4,000 modules can be loaded simultaneously) in the IDM builder process during loading, the IDM builder process is restarted and the message *"[I: restart build server]"* is output. Here, the loaded, reusable import modules are lost.

5.2.3 Usage Information for the IDM Eclipse Plugin

For the use of the builder process mode with the MAKEGEN PLUGIN, the options **-IDMconsole** and **+builder** should be used. They enable Eclipse CDT (C/C++ Development Tooling) for Makefile Projects, to check IDM error messages from the IDM or PIDM error parser of the IDM ECLIPSE PLUGIN.

5.2.4 Particularities

- If an IDM builder process has to be started first, this is followed by a short delay as the process start must be waited for.
- During the build run, waiting times may arise that are caused by the delayed completion of the builder following a time-out or by the make process waiting for the termination of all child processes or the closing of the output channels. In this case, the IDM builder process can simply be stopped by calling IDM or PIDM with the **-builderstop** option.
- Communication between the IDM or the PIDM in the builder process client server mode is performed via a **named pipe** (on UNIX, typically in the */tmp* directory). The respective safety settings and access authorizations should also be made. A forced termination of IDM or PIDM – e.g. with `kill -9` – should be avoided in order to ensure the cleanup of the pipe files.
- When using **+builder** the forwarding of the output of the trace or the log file (e.g. via the options **-IDMerrfile** and **-IDMtracefile**) is not effective as the IDM build process is started without redirection.
- On MICROSOFT WINDOWS the error messages of the IDM or PIDM are normally displayed as messageboxes and other outputs are written to the IDM log file. To receive error messages in one console, the option **-IDMconsole** may be used. This option is passed on when starting the IDM builder process. This should enable the build processes to pass on its relevant error messages and outputs to **stdout** or **stderr**.

5.3 Command Line Options for the Builder Process

IDM and PIDM

- **+/-builder**
  Starts the IDM or PIDM in builder mode
- **-builderid <builder identifier>**
  Indicates an identifier for the IDM builder process
- **-builderstop**
  Stops the IDM builder process
- **-buildertimeout <secs>**
  Waiting time before the IDM builder process stops

See

Chapter “Command Line Options in the Simulation Program” for descriptions of the options.

IDM Library

- **-IDMconsole** (MICROSOFT WINDOWS)
  Redirects log file and error messages to STDOUT

See

Chapter “Command Line Options” for the description of the option.

6 Error messages of IDM

Error messages from the IDM are typically output in a dialog box in Windows, but in a log- or trace-file in UNIX/Linux. Although the **-IDMerrwinfile** startup option can also be used under Windows to redirect to a file, if this option is not used it can lead to a large number of error messages, which usually tempts the user to “kill” the IDM application program. The error message dialog box has now been equipped with a Cancel button, so that further popping up of error messages in a dialog box is prevented. Instead, these messages are written to the log file as [E:...] messages. However, the application is not aborted, nor is it possible to abort the application, but only to prevent the dialog boxes from popping up. If desired, this can be done by the application programmer in an error handler. The loading of a dialog/module is treated as an atomic block. This means that the suppression of the dialog boxes is effective until the end of this block. In principle, it is not possible to suppress ASSFAIL dialog boxes.

7 Functions for Error Analysis

Availability

The functions for error analysis described in this chapter are available in the IDM versions A.05.01.g3 and A.05.01.h as well as from A.05.02.e

7.1 Overview

The purpose of the functions described here is to further assist the developer in the analysis and classification of errors in special situations such as: interpreter error messages, application crashes[1] or fatal error messages. The following table contains a rough classification of possible errors in relation to typical points of error and the likely impact of such errors. The effect that an error remains undetected, leads to a changed or faulty functionality or may be transferred onto another component, is not explicitly mentioned in the table.

Point of Error	Description	Impact
Rule Code	Error in rule code of an IDM application, e.g. invalid access to an attribute or object, type errors or uninitialized values.	*** Eval Error in log or trace file [E: …] or [W: …] messages, YE- error codes in trace file
Application Function	Several error types, e.g. access violations, damaged memory management that in some cases may show through their impacts on the rule processing or the IDM library.	eventual crash
IDM Library	Faulty conditions within the IDM are normally secured through an assertion. However, an error can have an effect on the rule processing or it can appear as an application error.	FATAL ERROR in log or trace file [E: …] or [W: ..] messages in trace file
External Functions	External, IDM-independent functions, e.g. in a different thread or different library, can contain errors which can then be brought into other areas via actions such as overwriting memory or faulty synchronization.	e.g. crash

[1]A crash in this sense is an unhandled or a non-interceptable error situation, i.e. invalid memory access, division through 0.

In the following further information pertaining to the new additions for easier error analysis and their purpose can be found.

Notes

» In principle the new additions are not meant to replace or substitute a debugger or a tool such as "Dr. Watson" on MICROSOFT WINDOWS. They are to be seen as add-ons in order to deliver meaningful information that cannot be delivered by the mentioned tools.

» If previous serious errors have already occurred, then the correctness of the given information cannot be completely guaranteed.

7.1.1 Safety Tracing

In order to use the tracing function within the IDM for situations, in which application errors occur only after a longer period of time, the safety tracing mode can be used.

Normally, if the tracing function is running alongside application use this can lead to a decrease in performance and to a very large, cumbersome trace file. In the safety mode the tracing takes place in a ring buffer with a limited number of lines and line lengths thatis kept in the main memory. The buffer is then written in the trace file only after an application has been ended or after an error event.

Further information about safety tracing can be found with the comand line options **-IDMstrace**, **-IDMstracefile** and **-IDMstraceopts** in chapter "Command Line Options" and in chapter "Safety Tracing".

7.1.2 Dumpstate

Without tracing, the user receives no clues when crashes, "FATAL ERROR" or "Error in Eval" messages occur in applications that use binary files and the IDM runtime library. The dumpstate reveals status information of the IDM specifics which include the callstack, the errorstack, events and hints pertaining to the number of IDM objects and the use of memory through the IDM.

The primary goal is to output as much information as possible when an error occurs in order to be able to properly assess the error and at the same time simplify the causal research. Only information that is known to the IDM can be gathered; no information pertaining to the application or foreign functions. The writing of the dumpstate can be programmatically achieved via the Rule Language or the DM interface function.

Important Note

Security-related information, e.g. passwords that are entered into a log-in window, should be deleted by the application immediately after they have been used so that they cannot be revealed through a dumpstate and as a result be made accessible to others.

Further information about dumpstate output can be found with the command line options **-IDMcallstack**, **-IDMdumpstate** and **-IDMdumpstateseverity** in chapter “Command Line Options” and in chapter “Dumpstate (Status Information)”.

7.2 Exception Catcher

In order to be able to write out the safety tracing as well as the dumpstate after a crash, a global exception catcher is registered for the application. The exception catcher passes on the exception in order to allow for its further usual processing through the operating system (core dump on UNIX/LINUX or the writing of a dump file on MICROSOFT WINDOWS via “Dr. Watson”).

Note

When installing own exception catchers, please ensure that they pass on the exceptions.

The registration of an exception catcher can be prevented through the command line option **-IDMcatchexceptions** (see chapter “Command Line Options”).

8 Dumpstate (Status Information)

Availability

The dumpstate output is available in the IDM versions A.05.01.g3 and A.05.01.h as well as from A.05.02.e.

The dumpstate is the status information of IDM-relevant information in order to simplify the error analysis within an IDM application. The content of the dumpstate is divided into different sections that are variable and that are adapted to the error situation. In addition, the dumpstate is influenced by the errors that have previously occurred. For example an unsuccessful memory allocation leads to information concerning the memory usage by the IDM in the next dumpstate output. If no IDM objects or identifiers can be created, then the utilization of IDM objects and identifiers is dumped.

The dumpstate information is always encased between *"*** DUMP STATE BEGIN ***"* and *"*** DUMP STATE END ***"* and can have the following sections, which are described in detail in the paragraphs below:

- PROCESS: Process and thread number, date/time.
- ERRORS: Complete content of set error codes.
- CALLSTACK: Contains rules, DM interface functions and application functions directly called by the IDM.
- THISEVENTS and EVENT QUEUE: Actual processed thisevent objects and their values as well as events that are still in the queue.
- USAGE: The number of created objects, modules and identifiers and the size of the memory that is used by the rule interpreter and for string transfer.
- MEMORY: Memory usage as far as it can be detected by the IDM.
- SLOTS: Hints about IDM objects that have not been correctly released.
- VISIBLE OBJECTS: A list of the visible objects and their respective values.

In order to keep the output to a minimum, this is usually displayed in a shortened form. Normally, a shortening of the IDM strings (in "…") to a maximum of 40 characters always occurs. Their entire length is attached in [ ]. Byte size information is given in kilo, mega or gigabytes (k/m/g).

The dumpstate is normally automatically written to the log file or the trace file when one of the following situations occurs. Hereby, the situation also influences the dumpstate content.

Error Type	Dumped Sections
The rule interpreter delivers an "EVAL ERROR".	Errorstack, Callstack, Events.
The IDM library recognizes an error of its own and delivers a "FATAL ERROR".	All.

Error Type	Dumped Sections
An catchable exception (e.g. access violation, stack overflow, division by zero) comes about and the IDM exception catcher is active.	All – in addition, the exception code (MICROSOFT WINDOWS) or the signal number (UNIX) is initially displayed. On UNIX since IDM version A.05.02.e no complete dumpstate is output after an INT signal anymore, but only the sections Stack, Errors, Process and Events.
Normal shutdown via **DM_ShutDown()** but information about errors are at hand.	According to the hints.

Important Note

Safety-related information, for example passwords that are typed in a log-in window, should be deleted after being used by the application, so that they cannot be output in a dumpstate and thus be exposed.

The writing of the dumpstate can explicitly take place via the builtin function **dumpstate()** or the interface function **DM_DumpState()**. In addition, it is also possible to influence the dumpstate output independent of the type of error or to define the type of error in which a dumpstate output takes place via the options **-IDMdumpstate** and **–IDMdumpstateseverity**.

Due to the amount of complex information that is gathered by the dumpstate, crashes within the dumpstate functionality can happen and cannot be intercepted.

The following contains a detailed description of the various sections that can be found in the dumpstate:

8.1 Process

This area contains the process ID, the number of the thread in which the dumpstate is called as well as the thread number of the IDM main thread. If the dumpstate is not called from the IDM main thread, then the following message is displayed:

```
ATTENTION: not in IDM main thread!
```

When this happens extreme care should be taken. This indicates either an application error, an error in an external function or the improper use of the IDM. Please note that the callstack only contains the call list of the IDM main thread!

```
PROCESS:
pid=2984, thread=4080, IDM-thread=4080, date=2009-11-10, time=16:20:38
```

8.2 Errors

Lists all currently set error codes. See also interface function DM_QueryError in manual “C Interface - Functions”.

```
ERRORS:
#0: IDM-E-UnkAttr: Attribute not available for this type of object
[object/thing:639]
```

8.3 Callstack

The callstack (call list) is the most important tool in order to recognize if the reason behind a system crash is due to an error that occurred in an application function, in a rule or method, in the IDM library or in an external function.

The callstack contains all calls of rules, methods, built-in functions, IDM interface functions as well as the application functions called by the IDM. Hereby, the parameter values are displayed. However, in order to minimize the effect on performance it is not possible to carry out a correct string coding in every situation.

In addition, the this-object, the file name in which the rule is found, for ASCII dialogs the start-line of the rule as well as a %-output, which delivers the approximate position in the intermediate code of the rule, is also included. For the first rule on the stack the local variables (locvars list) as well as the content of the value stack (valstack), which is necessary for the expression evaluation, is also listed.

```
STACK:
#0: rule D.AfterError, this=dialog D, file=dumping.dlg:68+39%
  locvars=[dialog D, "initvar2", nothing]
  valstack=[100]
#1: rule on dialog start, this=dialog D, file=dumping.dlg:78+35%
#2: DM_StartDialog(dialog D, null)
```

Abridgements

Only the first 20 and the last 20 entries are listed.

Important

If the message *“ATTENTION: not in IDM main thread!”* appears at the beginning of the dumpstate output, extreme care should be taken. The callstack only displays the callstack of the IDM main thread!

8.4 Events

In this section the actual processed events (THISEVENTS) and the events that are currently waiting in line (EVENT QUEUE) to be processed are listed.

The source indicates the event trigger: **dialog**, **setval**, **destroy** or **external**. This reveals whether this has to do with an event that was triggered by a user or the system, if it was triggered by a change in value of an attribute, if it has to do with the destruction of a module or if it has to do with an external event.

The object, which is the recipient of the event, is also shown. The event queue, the data value and the arguments as well as the specific attributes of THISEVENTS (e.g. *.x*, *.y*) are also displayed.

```
THISEVENTS:
#0: source=dialog, object=dialog D, event=start
EVENT QUEUE#1:
#0: source=dialog, object=window D.Wi, event=activate
#1: source=external, object=pushbutton D.Wi.Pb, data=1, args=["EvData"]
```

Abridgements

A maximum of 10 events are listed

8.5 Usage

This section is made up of three tables:

- Information concerning the number of IDM objects, their distribution on defaults, models and instances, their total number and their approximate memory need in the IDM core (not the displayed user interface elements). This table is sorted in an ascending order in the columns **alloc** and **count**.
- Information pertaining to dialogs and modules. The number of all dialogs and modules is displayed. Furthermore, the sum of the object slots contained in the dialogs and modules, the memory that is needed for the slot administration, the complete number of identifiers and the memory needed for this is also shown. In the **<maximum>** line the dialogs and modules with the largest values are listed in the individual columns. This makes the dialogs and modules, upon which no further objects can be created or whose space for identifiers is becoming scarce, easier to identify.
- The third table is of an informative nature and contains the number and the memory size of the rule frames that are used by the rule interpreter. It also contains information pertaining to the number and memory size of buffers that are necessary for the storage of interim results.

```
              class  defaults    models instances     count     alloc
------------------- --------- --------- --------- --------- ---------
        thisevclass         0         0         1         1        84
          clipboard         0         0         1         1       100
         setupclass         0         0         1         1       140
        accelerator         1         1         2         4       336
             module         0         0         1         1       628
         pushbutton         1         0         1         2       752
             dialog         0         0         1         1       783
             text         1         1         7         9       828
         edittext         1         0         1         2       980
           window         1         1         1         3        2k
             rule         2         3         5        10       19k
          <total>         7         6        22        35       25k

   module     count     slots     alloc    labels labelsize name
--------- --------- --------- --------- --------- --------- -----------------
--
   dialog         1        27        3k        29        3k
   module         1         8        3k         1        3k
<maximum>                  27        3k        29        3k dialog D

   frames   f-alloc   scratch   s-alloc    values   v-alloc
--------- --------- --------- --------- --------- ---------
        2       22k         4        4k        69       17k
```

The columns **values** and **v-alloc** of the third table are not output in the IDM versions A.05.01.g3 and A.05.01.h.

Abridgements

A maximum of 126 classes are listed. No object class-related allocation sizes are determined (column contains only zeros).

8.6 Memory

This section contains the size of the allocated heap storage and gives clues about the amount of memory used for the entire application. This functionality is only available under Windows or with an activated IDM memory debugging. The sum of the allocated memory blocks is determined via **HeapWalk()** and for the C-runtime via _heapwalk. The heap that is used directly by the IDM is marked with “*” and the default heap of the process is marked with “P”.

```
MEMORY:
        heap     alloc     other
- ---------- --------- ---------
*        crt      386k       56k
  0x02CC0000        4k       61k
  0x02BA0000        5k      167k
  0x029C0000      386k      613k
  0x02A10000        9k       56k
  0x006F0000        7k      999k
  0x00340000       12k       52k
  0x00630000       11k       53k
P 0x007B0000      137k      797k
     <total>      568k        3m
```

Abridgements

Only the first 20 heaps are displayed.

Note

Please be aware that the heaps also contain IDM-foreign memory allocations that are, for example, used by the application function, system routines or external functions (DLL, In-Process-OLE-Control).

8.7 Slots

The output of slots serves especially for the recognition of errors in the IDM library, which are related to the referencing, the release and the destruction of objects. A list of object slots, which could not be completely eliminated or that displayed suspiciously high referencing and locking counters, is created.

```
SLOTS:
        slot       class       refs locks drop hull object
------------ ----------- ---------- ----- ---- ---- --------------------
[0x00020028]      window          1     1    1    0 window D.WINDOW:[1]
```

Abridgements

A maximum of 20 slots is listed. All locked slots are displayed in full.

If the command line option, bulit-in or interface function with the parameter *dump_locked* (see command line option **-IDMdumpstate <enum>** in chapter “Command Line Options”) is used for the dumpstate output, then additionally all attributes of the locked objects are dumped. Attribute values of resources, rules and functions cannot be dumped.

8.8 Visible Objects

In this section all of the visible objects are written out including the values of their pre-defined attributes and visible child objects. This section should deliver copies of most of the relevant objects and attributes when display problems occur in order to be able to reproduce errors more quickly.

```
VISIBLE OBJECTS:
window Wi {
  .repos_id "";
  .userdata nothing;
  .visible true;
  .sensitive true;
  .navigable true;
  .fgc null;
  .bgc null;
  .font null;
:
}
```

In the *dump_full* and *dump_uservisible* mode (see command line option **-IDMdumpstate <enum>** in chapter “Command Line Options”) all visible objects that are directly attached under a dialog or a module are written out. All of the pre-defined and user-defined attributes including all of the visible and invisible child objects and child records are written out for these objects.

In principle no resources, rules, methods or functions can be dumped.

Abridgements

Only the top visible objects are dumped, without value/child objects.

8.9 Example

Constellation

An IDM application started an individual thread via C that crashed after 10 seconds because it was trying to access an invalid storage address. In the thread approximately 5 x 10MB memory is allocated. During this process the IDM is constantly creating windows and records whereby the windows are eventually destroyed, but not the records.

In this case the end of the log file looks somewhat like the following:

```
FATAL ERROR: Exiting due to exception 0xC0000005 (ACCESS VIOLATION)
*** DUMP STATE BEGIN ***

ATTENTION: not in IDM main thread!

PROCESS:
pid=3024, thread=5176, IDM-thread=4684, date=2009-11-16, time=16:58:52

STACK:
#0: create(window, dialog D, true)
#1: rule D.Test ("c-thread-access-violation"), this=dialog D,
file=bad.dlg:78+40%
  valstack=["c-thread-access-violation", window, true]
#2: rule on extevent 1 , this=dialog D, file=bad.dlg:112+71%
#3: DM_EventLoop(null)
THISEVENTS:
#0: source=external, object=dialog D, data=1

USAGE:
              class  defaults    models instances     count     alloc
------------------- --------- --------- --------- --------- ---------
             record         1         0     24704     24705         0
```

```
            window         1         0         1         2         0
         clipboard         0         0         1         1         0
            dialog         0         1         1         2         0
            module         0         0         1         1         0
        setupclass         0         0         1         1         0
       thisevclass         1         0         1         2         0
       accelerator         1         0         3         4         0
          function         0         0         6         6         0
              rule         1         1         5         7         0
              text         1         6        16        23         0
           <total>         6         8     24740     24754         0

   module     count     slots     alloc    labels labelsize name
--------- --------- --------- --------- --------- --------- -----------------
--
   dialog         1     24745       99k        28        3k
   module         1         9        3k         1        3k
<maximum>               24745       99k        28        3k dialog D

   frames   f-alloc   scratch   s-alloc
--------- --------- --------- ---------
        2       13k         5        5k

VISIBLE OBJECTS:
#0: window D.WiMain

MEMORY:
        heap     alloc     other
- ---------- --------- ---------
*        crt       11m        3m
  0x030D0000        4k       61k
  0x030E0000        5k      167k
  0x00300000       11k       54k
  0x02EF0000       11m        4m
  0x027A0000        6k       58k
  0x00030000       13k       52k
P 0x00A50000       48m      791k
     <total>       59m        5m

*** DUMP STATE END ***
```

The following can be read from the dumpstate output:

- The message *“ATTENTION: not in IDM main Thread!”* implies that the crash (**ACCESS VIOLATION**) does not take place in the IDM. For this reason the callstack is, in this situation, unsuspicious. When the crash occurred in another thread the IDM was calling the built-in function **create**.

- The number of ***record*** instances in the USAGE section is suspiciously high. This can be seen in the **record** line of the **class** table as well as in the **<maximum>** line of the **module** sub-table. In addition, the enormous storage need is reflected in the **crt** line of the MEMORY table.
- The MEMORY table shows that an enormous amount of memory (over 50 MB) has been accumulated in the default heap of the process (labeled as “P”).

8.10 Noteworthy for Windows/Threads

The IDM manages its own callstack for rule calls, methods, builtin functions, IDM interface functions and IDM application functions (see also chapter “Callstack”). For the IDM interface functions (except for **DM_SendEvent**, **DM_QueueExtEvent**, which **do not** appear on the callstack) a check takes place to see if they are called from the same thread in which the IDM was initialized. If this is not the case, then an error message is sent.

For the most part (with only a few exceptions) the IDM and its interfaces are not designed for use in multi-threading situations. The callstack only serves to manage IDM-specific functions and not for arbitrary application functions.

9 Tracing

With help of various command line switches, processing in the DM can automatically be debugged and tested.

9.1 Description of Tracing

With help of the tracing facility, the DM application can be debugged. The tracefile contains all executed rules, all called application functions and all called DM functions with their current parameters (please refer also to manual “C Interface - Basics”).

The following is also traced:

- built-in functions with side-effects (e.g. **create**),
- setvalues (always - not only when a *changed* event was created),
- call to format and canvas functions,
- DM-calls in network applications on the server part.

The general layout of the file is as follows:

```
[<Abbreviation>]<Action>
```

The explanation of what has currently be done is given in the <Action>.

The <Abbreviation> stands for the DM action. The following codes are possible:

[AC]	AppMain call. This is the call to the main user program called **AppMain**.
[AR]	AppMain return. This is the return value of **AppMain** to the DM.
[BA]	Builder action: Start of an action (e.g. **-writeexport**) in **builder process mode**.
[BC]	Built-in function call. A built-in function was called by the Dialog Manager.
[BD]	Builder return: End of an action (e.g. **-writeexport**) in **builder process mode**.
[BM]	Builder message: IDM message in **builder process mode**.
[BR]	Built-in function return. This is the return value of a built-in function.
[BX]	Builder command: Command for the IDM in **builder process mode**, e.g. **-buildertimeout**.
[DC]	Dump Call. Source code output of the active window triggered by pressing the function key defined with the command line option **-IDMobjdump_fkey**.

[DE]	Dialog event. This is a user event which was triggered by the user when interacting with the system.
[DR]	Dump Return. End of source code output of the active window triggered by pressing the function key defined with the command line option **-IDMobjdump_fkey**.
[DS]	Dialog start. This is the execution of the dialog start rule.
[EC]	Error Handler Call. An error handler is called.
[EE]	Exit. The application does not exit normally. The exit status is written into the trace file.
[EQ]	Return executing SQL statement. Return value after the execution of a SQL statement.
[ER]	Error Handler Return. Return from an error handler.
[EX]	External event.
[FA]	Function assignment phase. The parameters passed with write access (declared as output) are now assigned to their object attributes.
[FC]	Function call. A function of the application was called by the DM.
[FE]	Finish dialog. This is the dialog finish event which was triggered by calling the function **exit** in a rule.
[FR]	Function return. This is the return value of an application function
[IA]	Interface argument. This is an argument which was passed from the application to a DM function.
[IC]	Interface call. A DM function was called by the application.
[IE]	Interface error. The DM function returns an error.
[IR]	Interface return. This is the return value of a DM function.
[IV]	Interface value. Additional values passed on to or returned of interface functions..
[MC]	Method call. Call of a method.
[MR]	Method return. Return value of a method.
[NI]	Network information. This message describes which transport is tried out and which one was started.
[PD]	Finish of perform rule. The execution of a sub-rule is finished.
[PR]	Perform rule. This is the call of a sub-rule from another rule.
[RC]	Rule call. This is the call of a named rule.

[RR]	Rule return. This is the return value of a rule.
[SC]	Simulated function call.
[SE]	Setval event. This is an internal event which was triggered by setting an object attribute.
[SQ]	Execute SQL statement. Execution of a SQL statement.
[SR]	Simulated function return. Return value of a simulated function.
[SV]	SetValue. This a SetValue to an attribute of an object.
[TD]	Tracing disabled. Tracing is switched off.
[TE]	Tracing enabled. Tracing is switched on.
[UM]	User message. This message was written by the application with a call of **DM_TraceMessage**.
[VS]	Version string of the used DM version (5 lines).
[WE]	Window system event
[XD]	Finish execution of a rule.
[XR]	Execution rule. The execution of a rule is started.

9.2 Configuration of Tracing

If enormous dialogs are being processed the tracefiles increase accordingly. This is why the tracing can be configured. In addition the possibility to switch on and off the tracing (*setup.tracing := true/false*) is available.

The configuration of tracing enables you to switch off the tracing for special trace events. This is helpful for e.g. loops or other trace-intensive rules or program codes. You may switch off trace events at the Setup object. The attribute .tracing is indexed with a string:

```
setup.tracing["<abbreviation>"] := false;
```

Example

```
setup.tracing["RR"] := false;
```

This definition switches off all return values of named rules.

By switching off certain trace events – e.g. ["RC"] (call of a named rule) – you may also obtain a grouping effect: in the above example the calls of named rules are oppressed as well as the return values of named rules (see table below, column “Grouping”).

Note

Please note that the following trace events cannot be switched off:

- important trace events (see table below, column “Can be toggled by user”.)
- trace events via **DM_TraceMessage**

Warning

If you need IDM support, do not switch off too many trace messages, as we might not be able to help you efficiently. If important trace messages are missing it is not possible to get a satisfactory analysis.

Please not the following programming hints.

Right

```
variable boolean I_NEED_HELP_FROM_ISA_SUPPORT := true;
...
rule MyCode()
{
  TraceEvent("RC", false);
  ...
}

rule TraceEvent(string TraceTag, boolean Value)
{
  !! Check the global variable for support.
  if ( not I_NEED_HELP_FROM_ISA_SUPPORT ) then
    setup.tracing[TraceTag] := Value;
  endif
}
```

Wrong

```
rule MyCode()
{
  !! Do not look for help in your rules without a trace.
  !! Nobody knows which rule was called!
  setup.tracing["RC"] := false;
  ...
}
```

The following table summarizes how trace events are grouped and indented in the trace file. It also indicates whether the trace messages can be turned off by the user. In the table mean:

- “Abbreviation” of relevant trace event: Please refer to chapter “Description of Tracing”.
- “Grouping”: Trace event which also switches trace event in the first column.
- “Indention”:

=	no indention
>	to the right
<	to the left

Abbreviation	Grouping	Indention	Can be Toggled by User
IC	IC	>	yes
IR	IC	<	yes
IE	--	=	no
IA	IC	=	yes
IV	IC	=	yes
FC	FC	>	yes
FR	FC	<	yes
FA	FC	=	yes
EE	--	=	no
VS	--	=	no
UM	--	=	no
AC	AC	>	yes
AR	AC	<	yes
NI	--	=	no
TE	--	=	no
TD	--	=	no
SV	--	=	yes
PR	PR	>	yes
PD	PR	<	yes
SQ	SQ	>	yes
EQ	SQ	<	yes
BC	BC	>	yes

Abbreviation	Grouping	Indention	Can be Toggled by User
BR	BC	<	yes
RC	RC	>	yes
RR	RC	<	yes
MC	MC	>	yes
ME	ME	=	no
ML	ME	=	no
MR	MC	<	yes
DS	--	=	yes
FE	--	=	yes
SE	--	=	yes
EX	--	=	yes
DE	--	=	yes
XR	XR	>	yes
XD	XR	<	yes
SC	SC	>	yes
SR	SC	<	yes
WE	--	=	yes
EC	--	>	yes
ER	EC	<	yes

9.3 Time Marks during Tracing

The option **-IDMtracetime <No.>** is used to protocol the absolute or relative time needed for functions and rules during program run. It is thus possible to identify very time-consuming functions or rules for which tuning measures could then be taken .

There are three types of time measurement, which can be set using the *<No.>* parameter:

0

No times are logged in the trace file.

1

This value indicates start time mode. In this mode all start and end times are logged. The time needed for a single structure may then be calculated with the difference. In this mode only the system and user time will be considered.
The times are given in format [hh:mm:ss:uuu] at the beginning of line:

» hh = hours
» mm = minutes
» ss = seconds
» uuu = milliseconds

2

This value indicates the trace time mode. In this mode the time difference to the last logged call is given. It is thus possible to easily recognize how much time is needed for individual actions.
In this mode the time difference to the last trace output is given in the format [sss:uuu] at the beginning of line:

» ss = seconds
» uuu = milliseconds

3

This value specifies the real-time mode. In this case the real time is indicated for each action to be logged in the trace file.
In this mode the real time is given in format [hh:mm:ss] at the beginning of line:

» hh = hours
» mm = minutes
» ss = seconds

9.4 Safety Tracing

Availability

The safety mode of tracing is available in the IDM versions A.05.01.g3 and A.05.01.h as well as from A.05.02.e.

Safety tracing is a special trace file mode. In order to keep the trace file running during long application sessions without experiencing a slower running system and the use of too many resources, safety tracing uses a limited ring buffer that is held in the memory. For this use the option **-IDMstracefile <filepath>** instead of **-IDMtracefile <filepath>** or when using **-IDMtracefile <filepath>** switch on safety mode with the option **-IDMstrace**.

With respect to the ring buffer, limited means that the number of lines and the number of characters per line (without indents) is restricted. In addition, the length of the string values (typically displayed in "…") is shortened and the string length is attached in brackets [ ]. The limitations can be influenced via the option **-IDMstraceopts**. For the hierarchical indentation in the trace file two blank spaces are used. This **cannot** be influenced by the option **-IDMindent**. If the issued lines are discontiguous this is marked by a line and a colon. If the maximum length of the line is exceeded, this is marked with a tilde (~).

The content of the ring buffer is written to the trace file only after the application has been ended. It is necessary to have an active exception catcher in order to guarantee that files are saved in case of system crashes (see chapter “Exception Catcher”).

Switching off the tracing or the output of certain trace codes is **not possible** in the safety tracing.

The following limitations apply within the safety tracing:

	Minimum	Maximum	Default
Bytes per line	20	65,536	200
Lines	20	100,000	1,000
String length			40
Indent depth		50 (100 white spaces)	

Index

A

AC (AppMain call) 53
application 24
application layer 7
AppMain call 53
AppMain return 53
AR (AppMain return) 53
Auflösung 20

B

BA (Builder action) 53
BC (builtin function call) 53
BD (Builder return) 53
big-endian 24
BindFunctions 30
-bindir 24
BM (Builder message) 53
BOM 24
BR (builtin function return) 53
builder 24, 37-38
Builder action 53
Builder command 53
Builder message 53
builder process 36
- client 36-37
- command line options 39
- communication 37-38
- environment variables 37
- IDM Eclipse Plugin 38
- log file 38
- makefile 37
- MakeGen Plugin 38
- Microsoft Windows 38
- particularities 38
- process number 37
- server 36-37
- time-out 37
- tracing 38
- usage 37-38
- working directory 37

builder process mode 36
Builder return 53
builderid 25, 37
builderstop 25, 38
buildertimeout 25
builtin function call 53
builtin function return 53
BX (Builder command) 53
Byte Order Mark 24

C

callstack 12, 46
character encoding 23
classname 25
-cleancompile 25
cleancompile1 25
-cleancompile1 25
-cobbasename 26
-cobname 26

COBOL
- Unicode 28

code page 23
- encoding mark 23

command line option 12
- place holder 22

command line options
- builder process 39
- set code pages 23

command line switch 53

-compile 26

compile1 26

-compile1 26

config 33

configurable 33

configuration
- tracing 55

configuration file 12, 28, 33-34

copy file 32

-cpyname 27

D

DC (Dump call) 53

DE (Dialog event) 54

debugging 53

default
- object 7

definition part 7

development environment 8

dialog 7
- control 7
- event 54
- start 54

dialog layer 7

DM_SHARED_MODULES 36

DPI 17, 20

DR (Dump return) 54

Drag&Drop 18, 20

DS (dialog start) 54

dumpstate 13-14, 44
- callstack 46
- Errors 46
- Event Queue 46
- events 46
- Heap 48
- IDM main thread 45
- Memory 48
- Process 45
- sections 44
- Slots 49
- Stack 46
- Thisevents 46
- Usage 47
- Visible Objects 49

E

EC (Error Handler Call) 54

EE (Exit) 54

encoding mark 23

environment variable
- IDM_CONFIGFILE 12
- IDM_ERRWIN 16
- IDM_LOGFILE 15
- IDM_NO_YI_MONITORING 17

IDM_STRACEOPTS 19

IDM_TRACEFILE 20

place holder 22

environment variables 15, 35, 37

ER (ErrorHandler Return) 54

error file 15

error handler 54

Errors 46

EX (external event) 54

exception catcher 12

execute SQL statement 55

execution rule 55

exit 54

external event 54

F

FA (function assignment phase) 54

FC (Function call) 54

FE (Finish dialog) 54

file access 35

finish

dialog 54

execution rule 55

perform rule 54

FR (function return) 54

function

assignment phase 54

call 54

return 54

function prototype 30

function table 30

G

global variable 33

H

header file 30, 32

Heap 48

HighDPI 17, 20

I

IA (interface argument) 54

IC (interface call) 54

identifier 3

idl file 31

IDM builder process 36

IDM Eclipse Plugin 30

IDM_CONFIGFILE 12

IDM_ERRWIN 16

IDM_LOGFILE 15

IDM_NO_YI_MONITORING 17

IDM_SEARCHPATH 18

IDM_STRACEOPTS 19

IDM_TRACEFILE 20

IDMbinerror 12

IDMcallstack 12

IDMcatchexceptions 12

IDMcolor 12

IDMconfigfile 12

IDMconsole 13, 38

IDMcp_appl 23

IDMcp_format 23

IDMcp_input 23

IDMcp_io 23

IDMcursor 13

IDMdumpstate 13

IDMdumpstateseverity 14

IDMenv 15, 37

IDMerrfile 13, 15

IDMerrwinfile 15

-IDMfatalneterrors 16

IDMfont 16

IDMformat 16

IDMindent 16

IDMkeyboard 17

IDMlanguage 17

IDMno_yi_monitoring 17

-IDMobjdump_fkey 17

IDMscale 17

-IDMsearchpath 18

IDMserver 18

-IDMshowerror 18

IDMsource 18

IDMstrace 19

IDMstracefile 19

IDMstraceopts 19

IDMtarget 20

IDMtile 20

IDMtiledpi 20

IDMtracefile 20

IDMtracetime 20, 58

-IDMusepathmodifier 21

IDMversion 22

IE (interface error) 54

-ifdir 27

IMDcp_output 23

include file 30

indenting

  source code 16

  tracing 16

initialization 33

interface

  argument 54

  call 54

  return 54

Interface

  error 54

interface value 54

IR interface return 54

IV (interface value) 54

L

little-endian 24

M

make 36

makefile 37

MC (method call) 54

Memory 48

method call 54

method return 54

-mfviscob 27

-mfviscob-u 28

model 7, 31

MR (method return) 54

Muster 20

N

National Character 28

network

information 54

NI (network information) 54

-noif 28

O

object 7

OLE server 31

operation rule part 7

option

-bindir 24

-cleancompile 25

-cleancompile1 25

-cobbasename 26

-cobname 26

-compile 26

-compile1 26

-cpyname 27

-IDMfatalneterrors 16

-IDMobjdump_fkey 17

-IDMsearchpath 18

-IDMshowerror 18

-IDMusepathmodifier 21

-ifdir 27

-mfviscob 27

-mfviscob-u 28

-noif 28

-recompile 28

-recompile1 28

-ufcob 29

-userregistry 30

+writefuncmap 30

+writeheader 30

-writeole 31

+/-writetrampolin 32

application 24

bindir 24

builder 24

builderid 25

builderstop 25

buildertimeout 25

classname 25

cleancompile 25

cleancompile1 25

compile 26

compile1 26

IDMbinerror 12

IDMcallstack 12

IDMcatchexceptions 12

IDMcolor 12

IDMconfigfile 12

IDMconsole 13

IDMcursor 13

IDMdumpstate 13

IDMdumpstateseverity 14

IDMenv 15

IDMerrfile 15

IDMerrwinfile 15

IDMfont 16

IDMformat 16

IDMindent 16

    IDMkeyboard 17
    IDMlanguage 17
    IDMno_yi_monitoring 17
    IDMscale 17
    IDMsearchpath 18
    IDMserver 18
    IDMsource 18
    IDMstrace 19
    IDMstracefile 19
    IDMstraceopts 19
    IDMtarget 20
    IDMtile 20
    IDMtracefile 20
    IDMtracetime 20
    IDMusepathmodifier 21
    IDMversion 22
    ifdir 27
    noif 28
    profile 28
    recompile 28
    recompile1 28
    writeclassdef 30
    writedialog 30
Option
    -cleancompile1 25
    -compile1 26
    -recompile1 28
    cleancompile1 25
    compile1 26
    IDMtiledpi 20
    recompile1 28
options 8
Options in the Simulation Program 24

P

pattern 17
PD (finish of perform rule) 54
perform rule 54
    finish 54
place holder
    command line option 22
    environment variable 22
PR (perform rule) 54
presentation layer 7
Process 45
process number 37
profile 28, 33

R

RC (rule call) 54
RC (simulated function return) 55
-recompile 28
recompile1 28
-recompile1 28
record 32
    configurable 33
reg file 31
resource 7
Return executing SQL statement 54
RR (rule return) 55
rule 7
    call 54
    return 55

S

safety mode 19
safety tracing 19, 59
SC (simulated function call) 55
SE (setval-event) 55
search path 18
searchsymbol 29
setval-event 55
SetValue 55
shared modules mode 36
simulated function call 55
simulated function return 55
Slots 49
source code
  indenting 16
SQ (execute SQL statement) 55
Stack 46
status information 44, See also *dumpstate*
SV (SetValue) 55

T

TD (tracing disabled.) 55
TD (tracing enabled.) 55
testing 53
tile 17
Tile 20
time 20, 58
time-out 37
timestamp 20
trace file 20
tracefile 53
tracing 38
  description 53
  indenting 16
  safety mode 19, 59
tracing disabled 55
tracing enabled 55
tuning measures 58

U

-ufcob 29
UM (user message) 55
Unicode 28
Use Path 21
user message 55
-userregistry 30

V

variable
  configurable 33
version string 55
Visible Objects 49

W

WE (window system event) 55
windows system event 55
working directory 37
writebin 30
writeclassdef 30
writedialog 30
+writefuncmap 30
+writeheader 30
-writeole 31

writeproto 31

writerefs 31

+/-writetrampolin 32

X

XD (finish execution rule) 55

XR (execution rule) 55